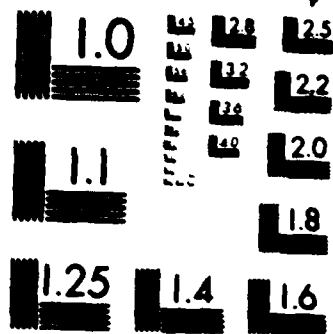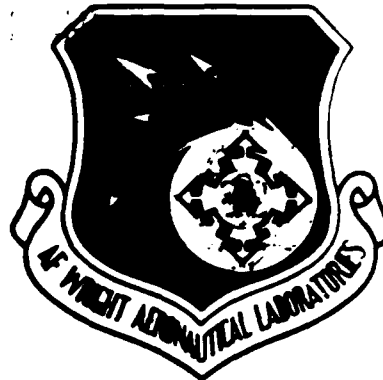END
8-87
DTIC

MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963-A

AD-A182 653

AFWAL-TR-86-4006
Volume VIII
Part 26

DTIC FILE

INTEGRATED INFORMATION
SUPPORT SYSTEM (IISS)
Volume VIII - User Interface Subsystem
Part 26 - Rapid Application Generator User Manual

General Electric Company
Production Resources Consulting
One River Road
Schenectady, New York 12345

Final Report for Period 22 September 1980 - 31 July 1985

November 1985

Approved for public release; distribution is unlimited.

DTIC
SELECTED
JUL 1 6 1987
E

MATERIALS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AFB, OH 45433-6533

H182 653

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
Unclassified	

2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
	Approved for public release;
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	distribution is unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
	AFWAL-TR-86-4006 Vol VIII. Part 26

6a. NAME OF PERFORMING ORGANIZATION	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
General Electric Company Production Resources Consulting		AFWAL/MLTC

6c. ADDRESS (City, State and ZIP Code)	7b. ADDRESS (City, State and ZIP Code)
1 River Road Schenectady, NY 12345	WPAFB, OH 45433-6533

8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
Materials Laboratory Air Force Systems Command, USAF	AFWAL/MLTC	F33615-80-C-5155

8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS.			
	PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
Wright-Patterson AFB, Ohio 45433	78011F	7500	62	01

11. TITLE (Include Security Classification)
(See Reverse)

12. PERSONAL AUTHOR(S)
Morenc, Carol and Stafford, Frances

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final Technical Report	22 Sept 1980 - 31 July 1985	1985 November	78

16. SUPPLEMENTARY NOTATION	The computer software contained herein are theoretical and/or references that in no way reflect Air Force-owned or -developed computer software.
ICAM Project Priority 6201	

17	COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	
1308	0905		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This manual describes the Application Definition Language and the process used for translating textual definitions of interactive database applications into programs that access selected database information resident in the Common Data Model (CDM). These data are accessible through the IISS Neutral Data Manipulation Language. To generate an application, you must first use the Application Definition language to define the application. You then go through several runtime steps to generate the executable application program from the application definition. This manual describes the syntax of the language and how to run the application generator and produce the executable.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
David L. Judson	513-255-6976	AFWAL/MLTC

11. Title

Integrated Information Support System (IISS)
Vol VIII - User Interface Subsystem
Part 26 - Rapid Application Generator User Manual

A S D 86 0040
9 Jan 1986

Accession For	
NTIS GRA&I	☒
DTIC TAB	☑
Unannounced	☐
Justification	

By
Distribution/

Availability Codes		
Dist	Avail and/or Special	
A-1		

PREFACE

This user's manual covers the work performed under Air Force Contract F33615-80-C-5155 (ICAM Project 6201). This contract is sponsored by the Materials Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Gerald C. Shumaker, ICAM Program Manager, Manufacturing Technology Division, through Project Manager, Mr. David Judson. The Prime Contractor was Production Resources Consulting of the General Electric Company, Schenectady, New York, under the direction of Mr. Alan Rubenstein. Tne General Electric Project Manager was Mr. Myron Hurlbut of Industrial Automation Systems Department, Albany, New York.

Certain work aimed at improving Test Bed Technology has been performed by other contracts with Project 6201 performing integrating functions. This work consisted of enhancements to Test Bed software and establishment and operation of Test Bed hardware and communications for developers and other users. Documentation relating to the Test Bed from all of these contractors and projects have been integrated under Project 6201 for publication and treatment as an integrated set of documents. The particular contributors to each document are noted on the Report Documentation Page (DD1473). A listing and description of the entire project documentation system and how they are related is contained in document FTR620100001, Project Overview.

The subcontractors and their contributing activities were as follows:

TASK 4.2

Subcontractors	Role
Boeing Military Aircraft Company (BMAC)	Reviewer.
D. Appleton Company (DACOM)	Responsible for IDEF support, state-of-the-art literature search.
General Dynamics/ Ft. Worth	Responsible for factory view function and information models.

Subcontractors	Role
Illinois Institute of Technology	Responsible for factory view function research (IITRI) and information models of small and medium-size business.
North American Rockwell	Reviewer.
Northrop Corporation	Responsible for factory view function and information models.
Pritsker and Associates	Responsible for IDEF2 support.
SofTech	Responsible for IDEF0 support.

TASKS 4.3 - 4.9 (TEST BED)

Subcontractors	Role
Boeing Military Aircraft Company (BMAC)	Responsible for consultation on applications of the technology and on IBM computer technology.
Computer Technology Associates (CTA)	Assisted in the areas of communications systems, system design and integration methodology, and design of the Network Transaction Manager.
Control Data Corporation (CDC)	Responsible for the Common Data Model (CDM) implementation and part of the CDM design (shared with DACOM).
D. Appleton Company (DACOM)	Responsible for the overall CDM Subsystem design integration and test plan, as well as part of the design of the CDM (shared with CDC). DACOM also developed the Integration Methodology and did the schema mappings for the Application Subsystems.

Subcontractors	Role
Digital Equipment Corporation (DEC)	Consulting and support of the performance testing and on DEC software and computer systems operation.
McDonnell Douglas Automation Company (McAuto)	Responsible for the support and enhancements to the Network Transaction Manager Subsystem during 1984/1985 period.
On-Line Software International (OSI)	Responsible for programming the Communications Subsystem on the IBM and for consulting on the IBM.
Rath and Strong Systems Products (RSSP) (In 1985 became McCormack & Dodge)	Responsible for assistance in the implementation and use of the MRP II package (PIOS) that they supplied.
SofTech, Inc.	Responsible for the design and implementation of the Network Transaction Manager (NTM) in 1981/1984 period.
Software Performance Engineering (SPE)	Responsible for directing the work on performance evaluation and analysis.
Structural Dynamics Research Corporation (SDRC)	Responsible for the User Interface and Virtual Terminal Interface Subsystems.

Other prime contractors under other projects who have contributed to Test Bed Technology, their contributing activities and responsible projects are as follows:

Contractors	ICAM Project	Contributing Activities
Boeing Military Aircraft Company (BMAC)	1701, 2201, 2202	Enhancements for IBM node use. Technology Transfer to Integrated Sheet Metal Center (ISMC).

v

Contractors	ICAM Project	Contributing Activities
Control Data Corporation (CDC)	1502, 1701	IISS enhancements to Common Data Model Processor (CDMP).
D. Appleton Company (DACOM)	1502	IISS enhancements to Integration Methodology.
General Electric	1502	Operation of the Test Bed and communications equipment.
Hughes Aircraft Company (HAC)	1701	Test Bed enhancements.
Structural Dynamics Research Corporation (SDRC)	1502, 1701, 1703	IISS enhancements to User Interface/Virtual Terminal Interface (UI/VTI).
Systran	1502	Test Bed enhancements. Operation of Test Bed.

TABLE OF CONTENTS

APPENDICES

FIGURES

SECTION 1

INTRODUCTION

The Rapid Application Generator (RAP) is a tool for translating any textual definition of an interactive database application into the C and COBOL programs that are required to access selected data base information resident in the Common Data Model (CDM). These data are accessible through the Integrated Information Support System (IISS) Neutral Data Manipulation Language (NDML). To generate an application, you must first use the Application Definition Language (ADL) to define the application. You then go through several steps defined in Appendix A to generate the executable application program from the application definition.

Manual Objectives

The main objective of this manual is to describe the syntax of the ADL. Topics covered include:

- How to define an application.

- The syntax of the language

- How to run the application generator and produce the executable.

Intended Audience

The RAP is intended for use by application programmers in the IISS environment. Knowledge of User Interface forms, the CDM, and the Neutral Data Manipulation Language (NDML) is assumed.

Related Documents

The Form Editor User Manual describes how to define the User Interface forms. Information you should be familiar with includes:

- item, form and window fields

- repeating fields

The Form Processor User Manual explains several concepts you need to help you in defining and using forms for application software. These include:

- form hierarchies

- qualified names

The Neutral Data Manipulation Language User Manual describes the syntax of the NDML. This language provides the capability of communicating with the IISS testbed databases. You need to be familiar with the SELECT, DELETE, INSERT, and MODIFY commands of the NDML.

SECTION 2

DOCUMENTS

2.1 Reference Documents

[1] Structural Dynamics Research Corporation, IISS Rapid
Application Generator Development Specification, DS
620144502 , 1 November 1985.

[2] Systran, ICAM Documentation Standards, 15 September
1983, IDS150120000C.

[4] Systran, User's Manual for the ICAM Integrated Support
System (IISS) Neutral Data Manipulation Language
(NDML), February, 1983.

[5] Systran, Implementation of Enhancements of NDML SELECT
COMMAND, 25 July 1984, revised 9 September 1984.

[6] Systran, Discussion of Function Implementation NDML
SELECT COMMAND, 25 July 1984, revised 4 September
1984.

This manual is one of a set of user manuals that together
describe how to operate in the IISS environment. The complete
set consists of the following manuals listed here for reference:

[1] Structural Dynamics Research Corporation, IISS Form
Editor User Manual, UM 6201444400B, 1 November 1985.

Explains how to define and maintain electronic forms.
It is intended to be used by programmers writing
application programs that use the Form Processor.

[2] Structural Dynamics Research Corporation, IISS Form
Processor User Manual, UM 620144200B, 1 November 1985.

Describes the set of callable execution time routines
available to an application program to process
electronic forms. It is intended to be used by
programmers writing application programs for the IISS
environment.

[3]   Structural Dynamics Research Corporation, IISS
      Terminal Operator Guide, OM 620144000 , 1 November
      1985.

      Explains how to operate the generic IISS terminal when
      running an IISS application program. The IISS end
      user environment, function selection and some
      predefined applications are also described.

[4]   Structural Dynamics Research Corporation, IISS Text
      Editor User Manual, UM 620144600B, 1 November 1985.

      Explains how to use the file editing functions
      including: inserting, deleting, moving and replacing
      text.

[5]   Structural Dynamics Research Corporation, IISS Rapid
      Application Generator User Manual, UM 620144502, 1
      November 1985.

      Describes the Application Definition Language and the
      process used for translating textual definitions of
      interactive database applications into programs that
      access selected data base information resident in the
      Common Data Model. This information is accessible
      through the IISS Neutral Data Manipulation Language.

[6]   Structural Dynamics Research Corporation, IISS Report
      Writer User Manual, UM 620144501, 1 November 1985.

      Describes the Report Definition Language and the
      process of creating a hard copy report of selected
      data base information resident in the Common Data
      Model. This information is accessible through the
      IISS Neutral Data Manipulation Language. This manual
      also describes the Hierarchical Report Writer which
      functions as a post processor to the initial report.

[7]   Structural Dynamics Research Corporation, IISS Virtual
      Terminal User Manual, UM 620144300B, 1 November 1985.

      Explains the program callable interface to the IISS
      Virtual Terminal. The callable routines, Virtual
      Terminal commands and the implementation of additional
      terminals are described. It is intended for
      application and system programmers working in the IISS
      environment.

2.2 Terms and Abbreviations

Application Definition Language an extension of the Forms Definition Language that includes retrieval of database information and conditional actions. It is used to define interactive application programs.

Application Generator (AG), subset of the IISS User Interface that consists of software modules that generate IISS application code and associated form definitions based on a language input. The part of the AG that generates report programs is called the Report Writer. The part of the AG that generates interactive applications is called the Rapid Application Generator.

Application Interface (AI), subset of the IISS User Interface that consists of the callable routines that are linked with applications that use the Form Processor or Virtual Terminal. The AI enables applications to be hosted on computers other than the host of the User Interface.

Application Process (AP), a cohesive unit of software that can be initiated as a unit to perform some function or functions.

Attribute field characteristic such as blinking, highlighted, black, etc. and various other combinations. Background attributes are defined for forms or windows only. Foreground attributes are defined for items. Attributes may be permanent, i.e., they remain the same unless changed by the application program, or they may be temporary, i.e., they remain in effect until the window is redisplayed.

Common Data Model (CDM), IISS subsystem that describes common data application process formats, form definitions, etc of the IISS and includes conceptual schema, external schemas, internal schemas, and schema transformation operators.

Communication Services allows on host interprocess communication and inter-host communication between the various Test Bed subsystems.

Communication Subsystem (COMM), IISS subsystem that provides communication services to the Test Bed and subsystems

Computer Program Configuration Item: (CPCI), an aggregation
of computer programs or any of their discrete portions, which
satisfies an end-use function.

Conceptual Schema: (CS), the standard definition used for
all data in the CDM. It is based on IDEF1 information
modelling.

Cursor Position: the position of the cursor after any
command is issued.

Device Drivers: (DD), software modules written to handle
I/O for a specific kind of terminal. The modules map terminal
specific commands and data to a neutral format. Device Drivers
are part of the UI Virtual Terminal.

Display List: is similar to the open list, except that it
contains only those forms that have been added to the screen and
are currently displayed on the screen.

External Schema: (ES), an application's view of the CDM's
conceptual schema.

Field: two dimensional space on a terminal screen.

Field Pointer: indicates the ITEM which contains the
current cursor position.

Form: structured view which may be imposed on windows or
other forms. A form is composed of fields. These fields may be
defined as forms, items, and windows.

Form Definition: (FD), forms definition language after
compilation. It is read at runtime by the Form Processor.

Forms Definition Language: (FDL), the language in which
electronic forms are defined.

Forms Driven Form Editor: (FDFE), subset of the FE which
consists of a forms driven application used to create Form
Definition files interactively.

Form Editor: (FE), subset of the IISS User Interface that
is used to create definitions of forms. The FE consists of the
Forms Driven Form Editor and the Forms Language Compiler.

Form Hierarchy: a graphic representation of the way in which forms, items and windows are related to their parent form.

Forms Language Compiler: (FLAN), subset of the FE that consists of a batch process that accepts a series of forms definition language statements and produces form definition files as output.

Form Processor: (FP), subset of the IISS User Interface that consists of a set of callable execution time routines available to an application program for form processing.

Form Processor Text Editor: (FPTE), subset of the Form Processor that consists of software modules that provide text editing capabilities to all users of applications that use the Form Processor.

IISS Function Screen: the first screen that is displayed after logon. It allows the user to specify the function he wants to access and the device type and device name on which he is working.

Integrated Information Support System: (IISS), a test computing environment used to investigate, demonstrate and test the concepts of information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous data bases supported by heterogeneous computers interconnected via a Local Area Network.

Item: non-decomposable area of a form in which hard-coded descriptive text may be placed and the only defined areas where user data may be input/output.

Logical Device: a conceptual device which to an application is indistinguishable from a physical device and is then mapped to part or all of a physical device.

Message: descriptive text which may be returned in the standard message line on the terminal screen. They are used to warn of errors or provide other user information.

Message Line: a line on the terminal screen that is used to display messages.

Network Transaction Manager: (NTM), IISS subsystem that
performs the coordination, communication and housekeeping
functions required to integrate the Application Processes and
System Services resident on the various hosts into a cohesive
system.

Neutral Data Manipulation Language: (NDML), the command
language by which the CDM is accessed for the purpose of
extracting, deleting, adding, or modifying data.

Open List: a list of all the forms that have been and are
currently open for an application process.

Operating System: (OS), software supplied with a computer
which allows it to supervise its own operations and manage
access to hardware facilities such as memory and peripherals.

Page: instance of forms in windows that are created
whenever a form is added to a window.

Paging and Scrolling: a method which allows a form to
contain more data than can be displayed with provisions for
viewing any portion of the data buffer.

Physical Device: a hardware terminal.

Presentation Schema: (PS), may be equivalent to a form. It
is the view presented to the user of the application.

Qualified Name: the name of a form, item or window preceded
by the hierarchy path so that it is uniquely identified.

Rapid Application Generator: (RAP), part of the Application
Generator that generates source code for interactive programs
based on a language input.

Subform: a form that is used within another form.

Text Editor: (TE), subset of the IISS User Interface that
consists of a file editor that is based on the text editing
functions built into the Form Processor.

User Data: data which is either input by the user or
output by the application programs to items.

User Interface: (UI), IISS subsystem that controls the user's terminal and interfaces with the rest of the system. The UI consists of two major subsystems: the User Interface Development System (UIDS) and the User Interface Management System (UIMS).

User Interface Development System: (UIDS), collection of IISS User Interface subsystems that are used by applications programmers as they develop IISS applications. The UIDS includes the Form Editor and the Application Generator.

User Interface Management System: (UIMS), the runtime UI. It consists of the Form Processor, Virtual Terminal, Application Interface, the User Interface Services and the Text Editor.

User Interface Monitor: (UIM), part of the Form Processor that handles messaging between the NTM and the UI. It also provides authorization checks and initiates applications.

User Interface Services: (UIS), subset of the IISS User Interface that consists of a package of routines that aid users in controlling their environment. It includes message management, change password, and application definition services.

User Interface/Virtual Terminal Interface: (UI/VTI), another name for the User Interface.

Virtual Terminal: (VT), subset of the IISS User Interface that performs the interfacing between different terminals and the UI. This is done by defining a specific set of terminal features and protocols which must be supported by the UI software which constitutes the virtual terminal definition. Specific terminals are then mapped against the virtual terminal software by specific software modules written for each type of real terminal supported.

Virtual Terminal Interface: (VTI), the callable interface to the VT.

Window: dynamic area of a terminal screen on which predefined forms may be placed at run time.

Window Manager: a facility which allows the following to be manipulated: size and location of windows, the device on which an application is running, the position of a form within a window. It is part of the Form Processor.

SECTION 3

APPLICATION DEFINITION LANGUAGE

The Application Definition Language (ADL) provides a very precise and flexible method for defining applications. It is an extension of the Form Definition Language that allows you to:

- Define the interactive environment that the application user will use to access the database (i.e., CDM data).

- Retrieve, delete, insert and modify the data in the database.

- Perform simple statistical calculations on the information such as counts, sums, and averages.

NOTE: Statistics are not available in Release 2.0.

3.1 <u>ADL Format Notation</u>

This manual uses the following notation to describe the syntax of the ADL.

UPPER-CASE	identifies reserved words that have specific meanings in the ADL. These words are generally required unless the portion of the statement containing them is itself optional.
lower-case	identifies names, numbers, or character strings that the user must supply.
Initial upper-case	identifies a statement or parameter that is defined later on.
_ Underscores	identify reserved words or portions of reserved words that are optional.
{ } Braces	enclosing vertically stacked options indicate that one of the enclosed options is required.
[ ] Brackets	indicate that the enclosed statement or parameter is optional. When two or more options are vertically stacked within the brackets, one or none of them may be specified.
... Ellipsis	indicates that the preceding statement or parameter may be repeated any number of times.

3.2 Application Definition Language Syntax

The complete ADL Syntax is listed in this section to show the structure of the language. The next section describes the characteristics and restrictions of the language statements.

Application Definition

CREATE APPLICATION application_name

    [Form_Definition] ...

    [Condition_Definition] ...

Form Definition

```
CREATE FORM form_name

     [ CONDITIONAL ]

     [ SIZE cols [ BY rows ] ]

     [ BACKGROUND { WHITE } ]
                  { BLACK }
     [ KEYPAD ( KEY n = function_name ... ) ]

     [ PROMPT Location prompt_string ] ...

     [ Field_Definition ] ...
```

Field Definition - Items

```
ITEM item_name [ Repeat_Spec ]

     [ Location ]

     [ SIZE cols [ BY rows ] ]

     [ VALUE {expression      } ]
             {INDEX(field_name)}
             {'._DATE'         }
             {'._TIME'         }

     [ NODUP ]

     DISPLAY AS { INPUT  }
                { OUTPUT }
                { HIDDEN }
                { TEXT   }

     [ DOMAIN ([LEFT ] [UPPER] [ PICTURE picture_spec ])]
              [RIGHT] [LOWER]

              [ MUST ENTER ] [ MUST FILL ] [ NUMERIC ]
              [ MAXIMUM int ] [ MINIMUM int ]

     [ PROMPT Location prompt_string ...]

     [ HELP { string    } ]
            {form_name  }
            {APPLICATION}
```

<u>Field Definition - Forms</u>

FORM form_name [Repeat Spec]

    Location

    SIZE cols [BY rows]

    [PROMPT Location prompt_string] ...

<u>Field Definition - Windows</u>

WINDOW window_name [Repeat Spec]

    Location

    SIZE int [BY int]

    BACKGROUND { BLACK }
                 { WHITE }

    [PROMPT Location prompt_string] ...

<u>Repeat Spec</u>

```
+--                                                    --+
| ({ * } { HORIZONTAL } [ WITH int SPACES ] [ , ... ] )  |
|  {int} { VERTICAL   }                                  |
+--                                                    --+
```

Location

```
            /                                                              \
           i { [ int ] { LEFT  } OF [ field_name ] }  +-                   i
           i {          { RIGHT }                   }  i AND                i
           i { COLUMN int                           }  +-                   i
           i                                                                i
           i        { [ int ] { BELOW } [ field_name ] }  -+               i
           i        {          { ABOVE }               }   i               i
           i        { ROW int                          }  -+               i
           i                                                                i
           i { [ int ] { ABOVE } [field_name ] }  +-                       i
           i {          { BELOW }               }   i AND                   i
           i { ROW int                          }  +-                       i
  [Rpt] AT <                                                                 >
           i        { [ int ] { RIGHT } OF [ field_name ] }  -+            i
           i        {          { LEFT  }                   }   i            i
           i        { COLUMN int                           }  -+            i
           i                                                                i
           i { [ int ] { LEFT  } OF [ Rpt OF ] [ field_name ]  }  i         i
           i {          { RIGHT }                              }  i         i
           i                                                                i
           i { [ int ] { ABOVE } [ Rpt OF ] [ field_name ] }              i
           i {          { BELOW }                          }              i
           i                                                                i
           i int-1 int-2 [RELATIVE TO [Rpt OF] [field_name] }              i
            \                                                              /
```

Rpt

```
    /                  \
   i  TOP LEFT         i
   i  TOP              i
   i  TOP RIGHT        i
   i  LEFT             i
   <  CENTER           >
   i  RIGHT            i
   i  BOTTOM LEFT      i
   i  BOTTOM           i
   i  BOTTOM RIGHT     i
    \                  /
```

Condition Definition

```
      /                                                              \
     I (OVERFLOW BY field_name) Condition_Action ...   I
     I (CHANGE item_name) Condition_Action ...          I
     I (item_field {= } value) Condition_Action ...     I
  ON <                {!=}                              >
     I [(PICK function_name) Condition_Action ...       I
     I [(PICK function_name & CURSOR IN field)          I
     I         Condition_Action ...                     I
     I (STARTUP) Condition_Action ...                   I
      \                                                              /
```

Condition Action

```
      /                               \
     I PRESENT form_name [IN window ]  I
     I SET item_name = {string }       I
     I                 {integer}       I
     I HELP { string      }            I
     I      { form_name }              I
     < Delete_action                   >
     I Insert_action                   I
     I Modify_action                   I
     I Select_action                   I
     I EXIT                            I
      \                               /
```

Delete Action

DELETE FROM table WHERE Predicate

Insert Action

INSERT INTO table (col_spec [...]) VALUES value_list

Modify Action

MODIFY table [USING table] SET column_assignment_list
     WHERE Predicate

Select Action

```
SELECT qualified_name = col_spec ...
       [ DISTINCT ]
       [ FROM table [ abbreviation ] .... ]
       [ WHERE Predicate ]
       [ ORDER BY Col_spec { ASCENDING } ... ]
                          { DESCENDING}
       [ '{' select_action ... '}' ]
```

Predicate

```
    /                        \
   | predicate AND predicate  |
   < Operand Operator Operand >
   |                          |
    \                        /
```

Operand

```
    /            \
   | Col_spec     |
   | string       |
   < number       >
   | field_name   |
   |              |
    \            /
```

Operator

```
    /    \
   |  =   |
   |  !=  |
   <  >   >
   |  >=  |
   |  <   |
   |  <=  |
    \    /
```

Col_spec

```
    /                        \
   | column                   |
   < table . column           >
   | abbreviation . column    |
    \                        /
```

3.2.1 Application Definition Syntax

This section describes the characteristics and restrictions of the language statements.

The collection of ADL statements that define an application is an application definition. An application definition is created by writing ADL statements directly to an ADL source file with any text editor you might use to prepare a program source file. The ADL source file is processed by the Rapid Application Generator to produce an interactive application executable. The Form and Condition definitions may be written in any order.

The syntax for an application definition is:

              CREATE APPLICATION application_name

              [Form_Definition] ...

              [Condition_Definition] ...

CREATE APPLICATION

              Every application definition must begin with the
              CREATE APPLICATION statement. This tells the
              compiler that what follows is an application
              definition.

              CREATE APPLICATION is the statement keyword.

              application_name   is a unique name of up to 10
                                 letters and/or numbers
                                 associated with each
                                 application. An
                                 application_name is required.
                                 It cannot begin with a number
                                 and cannot be the same as any
                                 form name in the application
                                 definition. It is incorporated
                                 by the Rapid Application
                                 Generator into the Application
                                 name which is used to invoke
                                 the executable at run time.
                                 The exact form of the
                                 Application name created is
                                 system dependent and therefore
                                 the length of application_name

may be further restricted by
the system you are running on.

Form_Definition

The Form_Definition portion(s) of an application
definition define screens which application users
will use to interact with the database. The
forms may also describe the placement and
formatting of retrieved data on the screen.
Three types of information can be displayed:
fixed textual information, database data and
input received from the user at the terminal.

Condition_Definition

The Condition_Definition portion(s) of an
application definition specify predefined actions
that will occur as the result of user interaction
with the software via the terminal. The
interactions include: cursor positioning,
function key selection, change of value in an
item field or the occurrence of certain values in
an item field. These user initiated events
determine the course of execution of the
software. The possible predefined actions that
can be triggered by a user initiated event
include: deletion of data in a table, insertion
of data into a table, modification of data in a
table, and selection of data.

NOTE: selection is available in release 2.0.
Deletion, insertion and modification are not
available in release 2.0.

3.2.2 Form Definition Syntax

Form definitions are used to define the physical layout of
the terminal screen. They display selected data and allow for
user interaction with the application.

The syntax for a form definition is:

CREATE FORM form_name

    [CONDITIONAL]

    [SIZE cols [BY rows]]

    [BACKGROUND (WHITE)]
               (BLACK)

    [KEYPAD (KEY n = function_name ...)]

    [PROMPT Location prompt_string] ...

    [Field_Definition] ...

CREATE FORM

> Every form definition must begin with the CREATE FORM statement. This tells the compiler that what follows is a form definition. The end of a form definition is signalled by one of the following:
>
> - another CREATE FORM statement
> - an ON condition statement (Condition_Definition)

> CREATE FORM   is the statement keyword.

> form_name   is a unique name of up to 10 letters, and/or numbers associated with each form. You will use this name in specifying the qualified names for SELECT actions and condition definitions. A form_name is required and cannot begin with a number. The length of the form name may be further restricted by the system on which you are running.

CONDITIONAL

> The CONDITIONAL clause allows you to specify that

the form only appears on the screen as a result of a PRESENT action of some ON condition.

CONDITIONAL   is the clause keyword.

SIZE   This clause determines the number of columns and rows the form will occupy when it is displayed. This clause is optional and if specified, when the form is displayed in a form or window field, the size of the form or window takes precedence. This means that the form will be "clipped" if the form size is larger than the size of the form or window it it displayed in. When the form size is smaller than the field size, the form will "grow" to fill the form or window field.

    SIZE   is the clause keyword.

    cols   is the width of the form. It is expressed as the number of columns it will occupy when displayed.

    BY   is a reserved word that must be included when rows is specified. There must be a space before and after this word when used.

    rows   is the height of the field. It is expressed as the number of rows it will occupy when displayed. This parameter is optional and defaults to one if not entered.

BACKGROUND

This clause allows you to define the background of the form. This is analogous to specifying what color of paper a paper form is printed on. This clause is optional. If it is omitted, the background of the form defaults to black.

    BACKGROUND   is the clause keyword

WHITE   displays black characters on an
                               opaque white background
                               (sometimes known as reverse
                               video).

BLACK   displays white characters on an
                               opaque black background.

KEYPAD

This clause allows you to give a name to one of the
terminal programmable function keys. This clause
is optional.

KEYPAD   is the clause keyword.

KEY n   is the programmable function
                               key. n can be any integer in
                               the set {0,4 through 20}

function_name   is the name you choose to
                               describe the function you are
                               assigning to the key. The name
                               can be a maximum of 10
                               alphabetic characters. You
                               will use this name in ON PICK
                               statements to specify the key
                               that determines the action.

PROMPT

> This clause allows you to specify fixed textual
> information such as titles and descriptions that
> will appear on the form. It is optional and may be
> repeated as many times as needed to specify the
> textual information.

> PROMPT   is the clause keyword.

> Location   describes where the text will
>   be positioned on the form. The
>   syntax is described in section
>   3.2.6 of this manual.

> prompt_string   is the textual information to
>   be displayed on the form and
>   must be enclosed in double
>   quotes ("text").

Field_Definition

> Field definitions specify the fields a form
> contains. Forms can contain item, form and window
> fields. The information required to define a field
> is different for each field type.

3.2.3   Field Definition Syntax - Items

An item field is one that holds data. The application user
can enter data into some item fields and the value entered may
be used to determine the execution path of the application
software. Item fields are also used to display database
information by specifying the appropriate item fields when
retrieving the information from the database. The syntax for an
item field definition is:

```
ITEM item_name [ Repeat_Spec ]

    [ Location ]

    [ SIZE cols [ BY rows ] ]

    [ VALUE {string_constant  } ]
            {INDEX(field_name)}
            {'._DATE'          }
            {'._TIME'          }

    [ NODUP ]

    DISPLAY AS { INPUT  }
               { OUTPUT }
               { HIDDEN }
               { TEXT   }

    [ DOMAIN ( [LEFT ] [UPPER] [ PICTURE picture_spec ])]
             [RIGHT] [LOWER]

             [ MUST ENTER ] [ MUST FILL ] [ NUMERIC ]
             [ MAXIMUM int ] [ MINIMUM int ]

    [ PROMPT Location prompt_string ...]

    [ HELP { string    } ]
           {form_name  }
           {APPLICATION}
```

ITEM

This statement specifies that the form contains a
data field.

ITEM	is the statement keyword.
item_name	is a unique name of up to 10 letters, numbers, and/or underscores. It is used to specify where information selected from the database is to be stored. It is also used in ON Condition statements to specify the field whose value

3-15

triggers the Condition action.
It is required and cannot begin
with a number.

Repeat_Spec   specifies that the item appears
on the form more than once. An
item may repeat indefinitely,
either horizontally or
vertically, with m spaces
between repetitions to form
rows or columns. A repeat
specification may be followed
by another repeat to form two
dimensional arrays of fields.
The syntax for this parameter
is described in section 3.2.7
of this manual.

Location

Location specifies where the first occurrence of
the item field will be positioned on the containing
form. When the item field is being used merely as
a place holder and is not to be displayed, Location
is not needed. The syntax for Location is
described in section 3.2.6 of this manual.

SIZE

The SIZE clause determines the area on the form
that each occurrence of the item occupies. This is
specified by the width and height. An item may not
overlap other fields or text and must be within the
boundaries of its containing form. If the VALUE
clause is included in the field definition, SIZE is
optional for item fields. In this case, the width
of the item defaults to the length of the string
constant for the VALUE clause and the height is
one. SIZE is also optional when the item field is
being used as a place holder for a value not to be
displayed.

SIZE   is the clause keyword.

cols   is the width of the field. It
                        is expressed as the number of
                        columns it occupies on the
                        form.

BY   is a reserved word that must be
                        included when rows is
                        specified.

rows   is the height of the field. It
                        is expressed as the number of
                        rows it occupies on the form.
                        It is optional and defaults to
                        one if not entered.

VALUE

The VALUE clause allows you to specify a default
value for the item field. This value is displayed
on the screen. The VALUE clause is optional. If
omitted the item is blank filled. If used to
specify a default value and the SIZE clause is
omitted from the field definition, the item is
assumed to be one dimensional and its length is the
number of characters in the string constant. When
storing the date and time, the size of the item
must be at least 9.

VALUE   is the clause keyword.

string_constant   is a character string enclosed
                        in double quotes ("default
                        value"). If the SIZE clause is
                        included in the field
                        definition, the length of the
                        string constant must be no more
                        than the total number of
                        characters specified by the
                        size. For example, if size is
                        4 by 3, then the string
                        constant should be no more than
                        12 characters long. When
                        entering default values for
                        multi-dimensional fields,
                        concatenate the values and they

will be split apart
appropriately to fill the
field. For example, if the
size is 4 by 3 and the string
constant is "****++++====", it
will be displayed as

```
****
++++
====
```

INDEX(field_name) specifies that the value for
this field is the name of the
first displayed element of the
array "field_name". The array
must be on the same form as
this item field and be enclosed
in single quotes (i.e.,
INDEX('myfield')). If
"myfield" is a two dimensional
array, an example index would
be "myfield(1,1)".

'._DATE' stores the current date in the
item field. The format of the
date is MM/DD/YY. The SIZE of
the field must be at least 9.

'._TIME' stores the current time in the
item field. The format of the
time is HH:MM:SS. The SIZE of
the field must be at least 9.

NODUP This clause optionally suppresses the output of
unchanged item field values. For example, if the
value of one item in an array is the same on
several records, the records will be printed but
only the first instance of the value will appear on
the report.

DISPLAY AS

The DISPLAY AS clause controls access to the field
and determines how the item value appears on the
screen. This clause is required for every item
field.

DISPLAY AS   is the clause keyword.

INPUT   means that the user may enter a value for this item and the value is echoed on the screen. The area where the user may type is highlighted on the form.

OUTPUT   means the value is data retrieved from the database and may not be entered by the user. It is displayed in bold type on screens that can display bold.

TEXT   is the same as OUTPUT but the value is not displayed in bold type.

HIDDEN   means that the user may enter a value for this item but the value is not echoed on the screen. This option is typically used for items of privileged information such as passwords. The area where the user may type is highlighted on the form.

DOMAIN

The DOMAIN clause allows you to reformat the value a user enters and specifies entry requirements and restrictions for the item field. This clause is optional. When included, the options may be entered in any order.

DOMAIN   is the clause keyword.

LEFT   is used to qualify input and output fields. It positions the value so that it begins in the leftmost position of the field. Any leading blanks are removed.

RIGHT   is used to qualify input and
                       output fields. It positions
                       the value so that it ends in
                       the rightmost position of the
                       field. The value is padded
                       with leading blanks.

If neither LEFT nor RIGHT is specified, the value
is stored in its original form.

UPPER   is used to qualify input and
                       output fields. It converts all
                       lower-case characters of a data
                       value to upper-case.

LOWER   is used to qualify input and
                       output fields. It converts all
                       upper-case characters of a data
                       value to lower-case.

If neither UPPER nor LOWER is specified, the data
value remains in its original form.

MUST ENTER   is used to qualify input
                       fields. It specifies that the
                       user is required to enter a
                       value for the item before the
                       form can be processed by the
                       application program.

MUST FILL   is used to qualify input
                       fields. It specifies that the
                       user is required to enter a
                       value for the item and it must
                       fill every position of the
                       item. This option is typically
                       used for items such as phone
                       numbers or social security
                       numbers.

NUMERIC   is used to qualify input
                       fields. It specifies that only
                       numbers will be accepted for
                       the field value. If this
                       option is not specified, all
                       alphanumeric characters are
                       accepted.

MAXIMUM int is used to qualify input
fields. It specifies that only
numeric values will be accepted
for the field and that the
highest value is the number
"int". The abbreviation "MAX"
may be used for this option.

MINIMUM int is used to qualify input
fields. It specifies that only
numeric values will be accepted
for the field and that the
lowest value is the number
"int". The abbreviation "MIN"
may be used for this option.

If either MAX or MIN is
specified, the field becomes
NUMERIC. Both the MAX and MIN
options may be specified to
define a range of acceptable
field values.

PICTURE is used to qualify output
fields. It allows you to
define a COBOL format as the
output format for the item data
value. This includes numeric
and alphanumeric specification,
leading zero suppression,
decimal placement. leading sign
indicators, currency symbols.
and placement of embedded
commas.

picture_spec is a COBOL picture
specification (enclosed in
double quotes) defining the
specific output format
desired.

PROMPT

The PROMPT clause allows you to specify textual information associated with the item field such as labels and instructions.

PROMPT   is the clause keyword.

Location   specifies where the information appears on the form. The syntax for Location is described in section 3.2.6.

prompt_string   is the textual information to be displayed on the form and must be enclosed in double quotes ("text").

HELP

The HELP clause allows you to specify help text that will be displayed during run time in response to a user request.

HELP   is the clause keyword.

string   is the text that you want to be displayed enclosed in double quotes.

form_name   is the name of the form that you want to be used to display the help text.

APPLICATION   is a reserved word that signifies how the processing of the ·HELP· key will be handled by the application.

3.2.4 Field Definition Syntax - Forms

Form fields are used to incorporate a subform within a
form. Form fields provide a convenient way to repeat groups of
information on the screen. The syntax for a form field
definition is:

FORM form_name [Repeat_Spec]

Location

SIZE cols [BY rows]

[PROMPT Location prompt_string] ...

FORM

      This statement specifies that you are incorporating
a form within the current form being defined. The
form specified by this statement must be defined
with its own form definition. The form definition
must be in the current ADL source file.

FORM	is the statement keyword.
form_name	is a unique name of up to 10 letters and/or numbers. This form_name is the one used on the CREATE FORM statement to define this form. A form name is required and cannot begin with a number.
Repeat_Spec	specifies that the form appears on its containing form more than once. A form may repeat indefinitely either horizontally or vertically, with m spaces between repetitions to create rows or columns. That repeat specification may then be repeated to create two dimensional arrays of subforms. The syntax for this is described in section 3.2.7 of this manual.

3-23

Location

Location specifies the position of the first
occurrence of the form. Repeating occurrences are
then positioned relative to this occurrence based
on the repeat specification. The Location syntax
is described in section 3.2.6 of this manual

SIZE

The SIZE clause determines the area on its
containing form that each occurrence of the subform
may occupy. The top left character of the
containing form becomes the origin of the first
occurrence of the subform.

SIZE	is the clause keyword.
cols	is the width of the field. It is expressed as the number of columns it occupies on the form.
BY	is a reserved word that must be included when rows is specified.
rows	is the height of the field. It is expressed as the number of rows it occupies on the form. Rows is optional and defaults to one if not entered.

For form fields, the SIZE clause specifies the
maximum space that this subform will occupy on its
containing form. When a form contains a repeating
field and the number of repetitions is unknown, you
can define the size of the form to be open ended in
the appropriate direction(s). This is done by
using an asterisk (*) as the value for cols and/or
rows. This means that some other factor such as
the overflow of a fixed size form at a higher level
will control the size of the openended form.

PROMPT

The PROMPT clause allows you to display fixed
textual information associated with the form field
such as labels and instructions.

PROMPT	is the clause keyword.
Location	specifies where to position the information on the form containing this form. The syntax for Location is described in section 3.2.6.
prompt_string	is the textual information to be displayed on the form. It must be enclosed in double quotes ("text").

3.2.5 Field Definition Syntax - Windows

Window fields are used as place holders on a form for subforms to be supplied by the Rapid Application Generator at run time. The syntax for a window field definition is:

WINDOW window_name [Repeat_Spec]

Location

SIZE cols [BY rows]

[PROMPT Location prompt_string ...]

WINDOW

> This statement specifies that you are defining a field on the form as a window. Its contents will be determined by the Rapid Application Generator at run time.

WINDOW	is the statement keyword.
window_name	is a unique name of up to 10 letters, numbers and/or underscores. This name cannot begin with a number or an underscore and is used by the Apliction Generator at run time to determine where to put subforms.
Repeat Spec	specifies that the window appears on the form more than once. A window may repeat, either horizontally or vertically, n times with m spaces between repetitions to form rows or columns. That repeat specification may then be repeated to form two dimensional arrays of fields.

Location

> Location specifies the position of the first

occurrence of the window on the form. Repeating
occurrences are then positioned relative to this
occurrence based on the repeat specification. The
Location syntax is described in section 3.2.6 of this
manual.

SIZE

The SIZE clause determines the area on the form that
each occurrence of the window may occupy. The top
left character of the window field becomes the origin
of any subforms it contains.

SIZE	is the clause keyword.
cols	is the width of the field. It is expressed as the number of columns it occupies on the form.
BY	is a reserved word that must be included if rows is specified.
rows	is the height of the field. It is expressed as the number of rows it occupies on the form. Rows is optional and defaults to one if not specified.

For window fields, the SIZE clause reserves the space
on the host form for subforms to be supplied at run
time. If a subform is bigger than the reserved
window, it will be "clipped".

PROMPT

The PROMPT clause allows you to specify information
associated with the field such as labels and
instructions.

PROMPT	is the clause keyword.
Location	specifies where to position the information on the form containing the field. The syntax for Location is described in section 3.2.6 of this manual.

prompt_string   is the information to be
displayed. It must be enclosed
in double quotes
("prompt_string").

3.2.6 Location Syntax

Location specifies where fixed textual information and
fields will be positioned on a form. Location may be absolute
with respect to the origin of the form or relative to fields on
the form. The origin of a form is at the top left corner with
rows being positive down and columns positive to the right.
Relative locations are especially useful for positioning text
that is associated with fields. The syntax for location is:

```
          /                                                          \
          |  { [ int ] { LEFT  } OF [ field_name ] }  +-             |
          |  {           { RIGHT }                 }  | AND          |
          |  { COLUMN int                          }  +-             |
          |                                                          |
          |        { [ int ] { BELOW } [ field_name ] } -+           |
          |        {         { ABOVE }                 } |           |
          |        { ROW int                           } -+          |
          |                                                          |
          |  { [ int ] { ABOVE } [field_name ] }  +-                 |
          |  {         { BELOW }                }  | AND             |
          |  { ROW int                         }  +-                 |
[Rpt] AT <                                                            >
          |        { [ int ] { RIGHT } OF [ field_name ] } -+        |
          |        {         { LEFT  }                    } |        |
          |        { COLUMN int                           } -+       |
          |                                                          |
          |  { [ int ] { LEFT  } OF [ Rpt OF ] [ field_name ] }  |
          |  {         { RIGHT }                               }  |
          |                                                          |
          |  { [ int ] { ABOVE } [ Rpt OF ] [ field_name ] }         |
          |  {         { BELOW }                           }         |
          |                                                          |
          |  int-1 int-2 [RELATIVE TO [Rpt OF] [field_name] }        |
          \                                                          /
```

When defining relative locations, field reference points
are used. Figure 3-1 shows the nine possible reference points
that a field can have.

Top Left   Top   Top Right

```
  _____
 | *               *                  * |
 |                                      |
 |                                      |
Left | *            *            * |  Right
 |              Center                  |
 |                                      |
 |                                      |
 | *_____*_____* |
```

Bottom Left   Bottom   Bottom Right
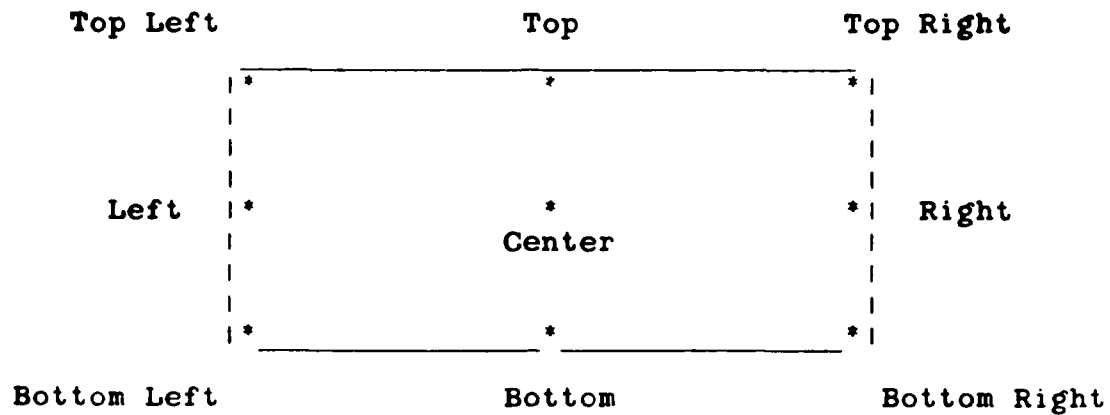
Figure 3-1   Field Reference Points

    Each of the reference points represents a character. This
means that if you have a one character field, all nine points
are the same.

    The syntax for the Rpt is:

```
    /                  \
   |  TOP LEFT         |
   |  TOP              |
   |  TOP RIGHT        |
   |  LEFT             |
   ‹  CENTER           ›
   |  RIGHT            |
   |  BOTTOM LEFT      |
   |  BOTTOM           |
   |  BOTTOM RIGHT     |
    \                  /
```

    When positioning text and fields, they must be contained
within the boundaries of the containing form and cannot overlap
other fields or text. Column one of a form must always be blank
and there must be one blank space between an item field and any
text. For example:

         text: _____

is legal and

         text:_____

is illegal.

     The following sections describe and show how to use the
location syntax to position fields. The syntax is basically the
same for positioning text. # represents the form origin and *
represents the default field reference point.

Absolute Location

              An absolute location positions the first character
              of a text string or a reference point of a field at
              the intersection of row (n) and column (m) with
              respect to the form origin. Both coordinates must
              be given. When positioning a field, the default
              reference point is the top left if the reference
              point is not given.

```
=======================================================
| | #                  ^
| |                    | n
| |                    |
| |                    v _____
| | <---------> | *                           |
| |      m      |                             |
| |            |       field_one             |
| |            |                             |
| |            |                             |
| |            | _____|
| |
| |
| |
```
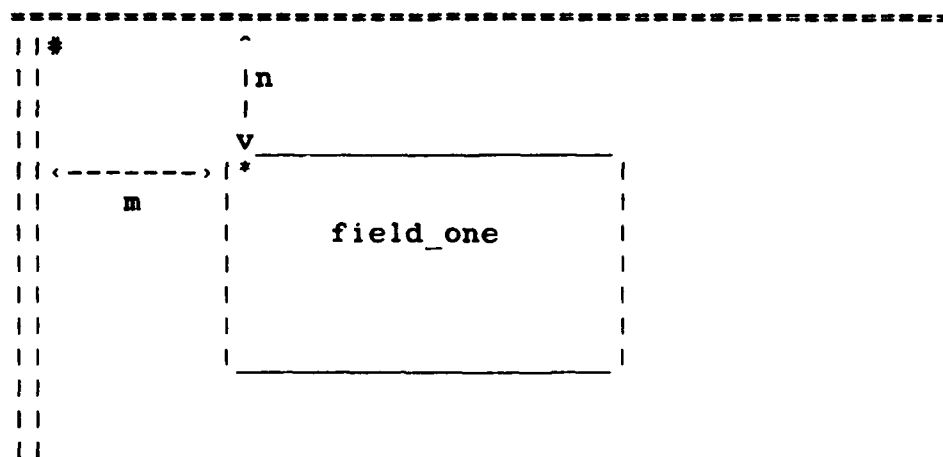
Figure 3-2   Absolute Location

              The reference point of field_one (*) is positioned
              at the intersection of row n and column m. To
              position field_one as shown, you would use the
              location syntax:

[Rpt] AT COLUMN int AND ROW int

or

[Rpt] AT ROW int AND COLUMN int

or

[Rpt] AT int-1 int-2 [RELATIVE TO [Rpt OF] field_name]]

where int-1 is the row and int-2 is the column. Some valid Locations are:

TOP LEFT AT COL m AND ROW n

TOP LEFT AT n m

AT n m

Relative Location

The reference point of a field can be positioned at
a point that is relative to the reference point of
another field on the form. If a specific reference
point for either of the fields is not given, the
default is the top left.

```
                              m
        <------------------------------> |*----------------------|
        ^                                |                        |
        |                                |                        |
        |                                |       field_two        |
        |                                |                        |
       |n                                |                        |
        |                                |_____|
        |
        v _____
        |*                   |
        |                    |
        |                    |
        |     field_one      |
        |                    |
        |                    |
        |_____|
```
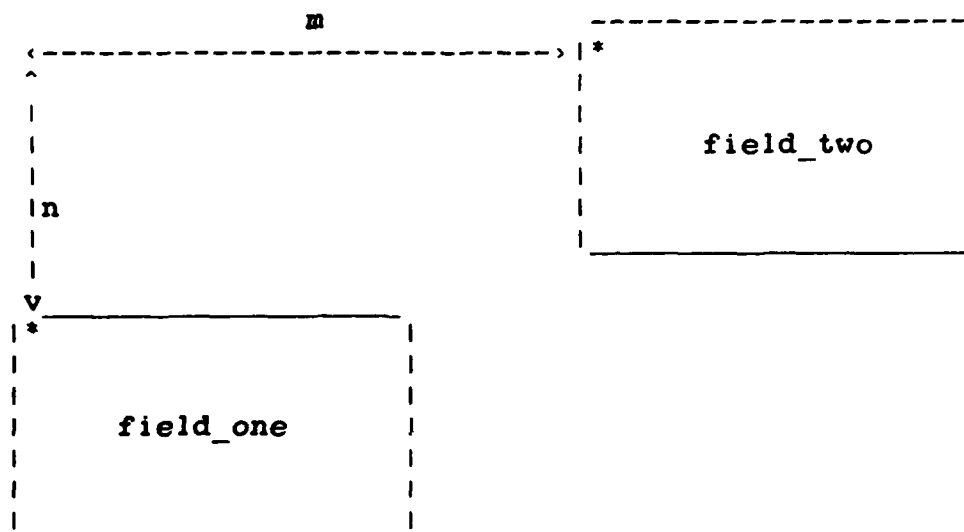
Figure 3-3   Relative Location

The reference point of field_two is positioned at row
n and column m relative to the reference point of
field_one. To position field_two as shown, you would
use the location syntax:

[Rpt] AT int-1 int-2

where int-1 is the row and int-2 is the column. Some
valid Locations are:

TOP LEFT AT -n m RELATIVE TO TOP LEFT OF field_one

AT -n m RELATIVE TO field_one

Relative Location (Above/Below)

The reference point of a field can be positioned n
rows above or below the reference point of another
field on the form. If reference points are not
given for either of the fields, the default
reference points are as shown by the *'s.

```
 _____
|                              |
|                              |
|                              |
|         field_one            |
|                              |
|_____*_____|
              ^
              |
              |n
              |
              |
              v
 _____*_____
|                              |
|                              |
|         field_two            |
|                              |
|_____|
```

Figure 3-4   Relative Location (Above/Below)

The reference point of field_two is positioned n
rows below the reference point of field_one. To
position field_two as shown, you would use the
location syntax:

[Rpt] AT [int] { ABOVE } [Rpt OF] [field_name]
                 { BELOW }

Some valid Location clauses are:

TOP AT n BELOW BOTTOM OF field_one

AT n BELOW field_one

Using the ABOVE keyword, you can position the
reference point of field_one n rows above the
reference point of field_two. Some valid Locations
are:

BOTTOM AT n ABOVE TOP OF field_two

AT n ABOVE field_two

Relative Location (Right/Left)

The reference point of a field can be positioned m
columns right or left of the reference point of
another field on the form. If reference points are
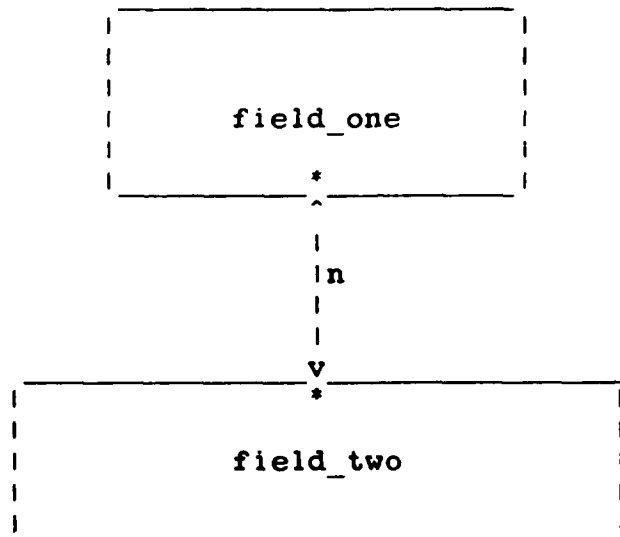not given for either of the fields, the default
points are as shown by the *'s.

```
 _____                          _____
|                        |                        |                          |
|                        |                        |                          |
|                        |              m         |                          |
|                        |                        |                          |
|    Field_one         * | <----------> |  *    Field_two                    |
|                        |                        |                          |
|                        |                        |_____|
|                        |
|                        |
|_____|
```
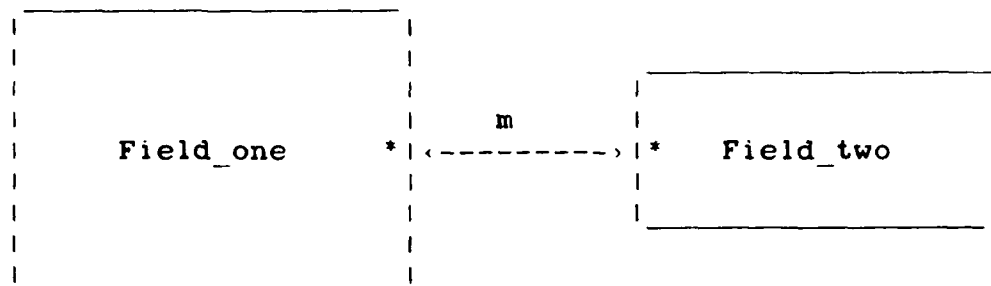
Figure 3-5   Relative Location (Right/Left)

The reference point (Rpt) of field_one is m columns
to the left of the field_two reference point or the
field_two reference point is m columns to the right
of the field_one reference point. To position the
fields as shown, you would use the location syntax:

[Rpt] AT [int] { LEFT } OF [Rpt OF] [field_name]
                 { RIGHT}

Some valid Locations are:

RIGHT AT m LEFT OF LEFT OF field_two

AT m LEFT OF LEFT OF field_two

LEFT AT m RIGHT OF RIGHT OF field_one

AT m RIGHT OF RIGHT OF field_one

AT m RIGHT OF field_one

NOTE: The Rpt is optional so locations 1 and 2 shown
above are identical as are locations 3 and 4.

Location Relative to Two Fields

> The reference point of a field can be positioned n
> rows above or below the default reference point of
> one field and m columns right or left of the
> default reference point of another field. Any
> reference point can be specified for the field
> being positioned. The reference points for the
> other fields default to the edge of the field
> closest to the field being positioned as shown by
> the *'s.

```
 _____
|                      |
|                      |
|    field_one         |
|                      |
|_____*_____|  _ _ _ _ _ _
                      |              ^
                      |              |          _____
                      |              |         |                        |
                      |              |         |                        |
                      |             |n         |*   field_two           |
                      |              |         |                        |
                      |              |         |_____|
                      |              |
 _____v_            |
|                     |* |< - - - - - - - |
|                     |     |            |
|   field_three       |     ■
|                     |
|_____|
```

Figure 3-6   Location Relative to Two Fields

The reference point of field_three is positioned n
rows below the bottom edge of field_one and m
columns left of the left edge of field_two To
position field_three as shown, you would use the
location syntax:

[Rpt] AT [int] (ABOVE) [field_name]
                 (BELOW)

AND [int] (LEFT) OF [field_name]

(RIGHT)

It does not matter which direction is specified
first. Some valid Locations are:

TOP RIGHT AT n BELOW field_one AND m LEFT OF
field_two

AT m LEFT field_two AND n BELOW field_one

AT n BELOW field_one AND m LEFT field_two

Combination Location

Field positions can be a combination of absolute
and relative locations. The reference point of a
field can be positioned at row n and m columns
right or left of the default reference point of
another field or at column m and n rows above or
below the default reference point of another field.
Any reference point can be specified for the field
being positioned. The reference point for the
other field defaults to the edge of the field
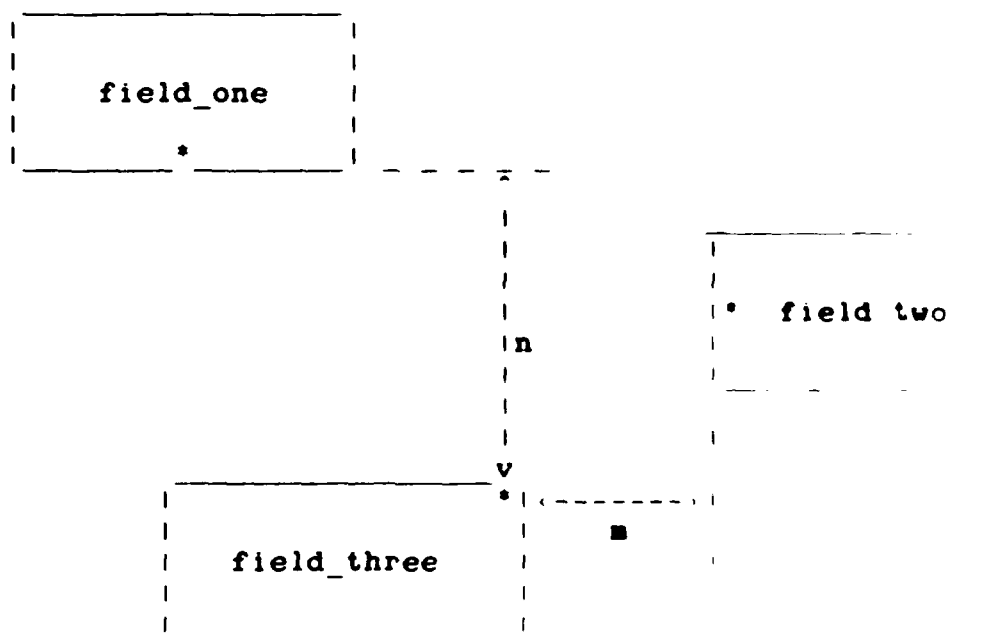closest to the field or text being positioned as
shown by the *'s.

```
=========================================================
                          ^
                          |
                          |              _____
                          |             |                |
                          |             |                |
                          |             |*    field_one   |
                          |n            |                |
                          |             |_____|
                          |
                          |                    |
                          v
  _____       *|<------------>|
 |                |       |         m
 |                |       |
 |   field_two    |
 |                |
 |_____|
```
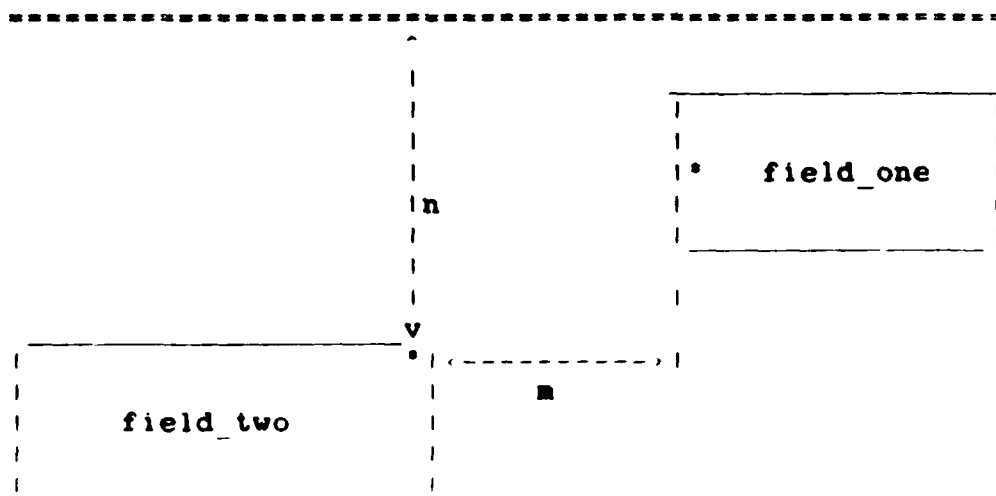
Figure 3-7   Absolute Row/Relative Column

The field_two reference point is in row n, m
columns left of the left edge of field_one. To
position field_two as shown, you would use the
location syntax:

[Rpt] AT ROW int AND [int] (LEFT) OF [field_name]
                              (RIGHT)

or

[Rpt] AT [int] { LEFT } OF [field_name] AND ROW int
                 { RIGHT}

Some valid Locations are:

TOP RIGHT AT ROW n AND m LEFT OF field_one

AT m LEFT field_one AND ROW n

AT ROW n AND m LEFT field_one
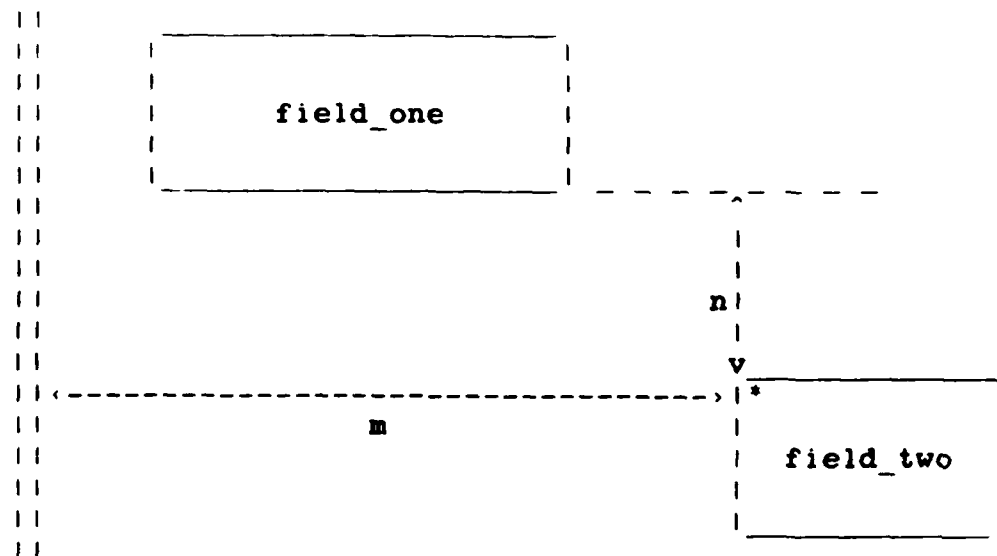
Figure 3-8 Absolute Column/Relative Row

The field_two reference point is in column m, n
rows below the bottom edge of field_one. To
position field_two as shown, you would use the
location syntax:

[Rpt] AT [int] {ABOVE} [field_name] AND COLUMN int
                 {BELOW}

or

[RPT] AT COLUMN int AND [int { ABOVE } [field_name]
                                { BELOW }

Some valid locations are:

TOP LEFT AT n BELOW field_one AND COL m

AT TOP LEFT n BELOW field_one AND COL m

AT COL m AND n BELOW field_one

3.2.7 Repeat Spec Syntax

The repeat specification specifies that the field appears
on the form more than once. A field may repeat, either
horizontally or vertically, an indefinite number times with m
spaces between repetitions to form rows or columns. The repeat
specification may then be repeated to form two dimensional
Carrays of fields. It is syntactically correct for a fixed size
form to contain a repeating field. The overflow of the fixed
size form can then be used to control paging and other
formatting of the display. The syntax for the Repeat
Specification is:

```
+--                                                        --+
| ({ int-1       } { HORIZONTAL } [ WITH int SPACES ] [....] )|
|  { int-1/int-2 } { VERTICAL   }                            |
|  { int-1/int-1 }                                          |
|  { *           }                                          |
|  { int-1/ *    }                                          |
+--                                                        --+
```

int-1   is how many times to repeat the field on the
                    form (i.e., display size).

int-1/int-2   indicates that the field is scrollable using
                    the function keys in the "scrll/page" mode of
                    the keyboard. Int-1 is how many times to
                    repeat the field on the form and int-2 is the
                    actual number of times the field occurs.

int-1/int-1   indicates that the field is scrollable using
                    the function keys in the "scrll/page" mode of
                    the keyboard but the keys are passed back to
                    the application program. Int-1 is how many
                    times to repeat the field on the form

*   specifies that the field repeats indefinitely
                    on the form.

int-1/ *   indicates a scrollable open-ended array which
                    is not yet supported.

HORIZONTAL   states that the field repeats in a horizontal
                    direction   The abbreviation H may be used for
                    this option.

VERTICAL   states that the field repeats in a vertical

direction. The abbreviation V may be used for
this option.

WITH   is an optional reserved word that may be
                 included for readability.

int-3   is how many spaces to leave between field
                 occurrences. It defaults to one if omitted.

SPACES   is an optional reserved word that may be
                 included for readability.

Row of Fields

>The Repeat Spec (* H 1) repeats a field
>horizontally with one space between repetitions.
>This would appear on the report as shown in
>Figure 3-9.

```
+---+   +---+                        +---+
|////|   |////|                        |////|
|////|   |////| . . . . . . . .        |////|
+---+   +---+                        +---+
```

Figure 3-9   Row of Fields

Array of Fields

>The Repeat Spec (* V 1,* H 1) repeats the row of
>fields defined by '* H 1' vertically with one blank
>row between repetitions. This would appear on the
>report as shown in Figure 3-10.

```
+---+   +---+                        +---+
|////|   |////|                        |////|
|////|   |////| . . . . . . . . .      |////|
+---+   +---+                        +---+

+---+   +---+                        +---+
|////|   |////|                        |////|
|////|   |////| . . . . . . . . .      |////|
+---+   +---+                        +---+

                  .
                  .
                  .
```

Figure 3-10   Array of Fields

3.2.8 Condition Definition Syntax

The Condition_Definition portion(s) of an application
definition specify predefined actions that will occur as the
result of user interaction with the software via the terminal.
The interactions include: cursor positioning, function key
selection, change of value in an item field or the occurrence of
certain values in an item field. These user initiated events
determine the course of execution of the software. The possible
predefined actions that can be triggered by a user initiated
event include: insertion of values into item fields,
presentation of other forms, deletion of data in a table,
insertion of data into a table, modification of data in a table,
and selection of data from a table.

NOTE: Selection is available in release 2.0. Deletion,
insertion and modification are not available in release 2.0.

The syntax for the Condition Definition is:

```
          /                                                    \
          | (OVERFLOW BY field_name) Condition_Action ...      |
          | (CHANGE item_name) Condition_Action ...            |
          | (item_field {= } value) Condition_Action ...       |
     ON   <             {!=}                                    >
          | [(PICK function_name) Condition_Action ...         |
          | [(PICK function_name & CURSOR IN field)            |
          |        Condition_Action ...                        |
          | (STARTUP) Condition_Action ...                     |
          \                                                    /
```

ON   is the statement keyword.

OVERFLOW Condition

This ON condition allows you to specify one or more
actions to be performed when the size of a field is
exceeded. A repeating field can cause the overflow
of its containing field either by repetition in the
horizontal or vertical direction. You specify the
name of the field that causes the condition action
to be triggered. For example, a set of nested
forms with a one row repeating item field at the
lowest level could be output with interruptions at
the line level or at a higher level. This allows
the application to require that a subform be output
either in its entirety or not at all.

OVERFLOW   specifies that one or more
                           actions will occur if the size
                           of a named field is exceeded.

BY   is a reserved word that must be
                           included.

field_name   is the qualified name of the
                           item or form that will trigger
                           the overflow if its next
                           occurrence causes the size of
                           its containing field to be
                           exceeded. It is a required
                           parameter. The name must be
                           qualified with enough of the
                           path name to make it unique.

Condition_action   specifies the action(s) to be
                           performed when the overflow
                           occurs. The syntax for this is
                           defined in section 3.2.9 of
                           this manual.

CHANGE Condition

This ON condition allows you to define one or more
actions to be performed when the value of a named
item field changes. If the item field being tested
for a changing value contains database values, then
it may be appropriate for these values to have been
previously sorted so that all similar values are
grouped together. This sorting can be achieved by
using the "ORDER BY" option of the SELECT statement
(see section 3.2.9 in this manual). If the test
for a changing value is positive, the actions are
taken after the item field is instantiated with the
changed value.

CHANGE   specifies that one or more
                           actions will be performed when
                           the value of a named item field
                           changes.

item_name   is the name of the item field
                           whose change of value will
                           trigger one or more actions.

Condition_action	specifies the action(s) to be performed when the item value changes. The syntax for this is defined in section 3.2.9 of this manual.

item_field { = } value Condition
        { != }

This ON condition allows you to define one or more actions to be performed when the value of a named item field is equal (not equal) to a value you specify. If the test for the value is positive, the actions are taken before the item field is instantiated with the value.

item_field	is the name of the item field whose value will be tested.
value	is the value of the item field that will cause the specified actions to be taken.
Condition_action	specifies the action(s) to be performed when the item value is equal to the value you specify. The syntax for this is defined in the section 3.2.9 of this manual.

PICK Condition

This ON condition allows you to define one or more actions to be performed when the user presses a programmable function key which you defined in the form definition KEYPAD clause. This condition can be used in combination with the CURSOR IN condition. If you use both conditions to trigger the actions, enter an ampersand "&" between them.

PICK	specifies that one or more actions will be performed when the user presses the specified key.

function_name   is the name you gave to the
                       programmable function key in
                       the form definition KEYPAD
                       clause.

CURSOR IN Condition

This ON condition allows you to qualify a PICK
condition by defining that the condition action(s)
be performed when the user positions the cursor in
a specified item or form field before pressing the
key specified in PICK. This condition can only be
used in combination with the PICK condition. If
you use both conditions, enter an & between them.

CURSOR IN   specifies that one or more
                        actions will be performed when
                        the user positions the cursor
                        in the defined location.

field   is the name of the field that
                        the user must position the
                        cursor in to trigger the
                        action(s).

Condition_action   specifies the action(s) to be
                        performed when the cursor is
                        positioned in the field. The
                        syntax for this is defined in
                        section 3.2.9 of this manual.

STARTUP Condition

This ON condition allows you to define one or more
actions to be performed at the beginning of the
application. This condition is required and should
include as one of its actions the presentation of
an initial form.

STARTUP   specifies that when the
                        application is started, one or
                        more actions are to be
                        performed.

Condition_action   specifies the action(s) to be
performed when the application
is started. The syntax for
this is defined in section
3.2.9 of this manual.

3.2.9 Condition Action Syntax

These are actions to be taken when the specified ON
condition occurs. The condition action syntax is:

```
    /                                   \
    |  PRESENT form_name [IN window ]   |
    |  SET item_name = {string }        |
    |                  {integer}        |
    |  HELP { string     }              |
    |       { form_name }               |
    .  Delete_action                    .
    |  Insert_action                    |
    |  Modify_action                    |
    |  Select_action                    |
    |  EXIT                             |
    \                                   /
```

PRESENT Action

This action causes the output of a specified form
or array element.

PRESENT   is the action keyword.

form name   is the name of the form field
                        to be output when the specified
                        ON condition occurs. This is a
                        required parameter.

IN window   allows you to output the form in
                        a specific window. If the window
                        is unspecified, then 'screen' is
                        assumed.

SET Action

This action sets the value of an item field to a
specified value   The value is either a constant or
the result of an expression which is evaluated
before assignment to the item.

SET   is the action keyword.

item_name   is the name of the item whose
                        value you want to set. It is
                        required and may be a qualified
                        name in order to uniquely
                        identify the item.

constant   the value. Either an integer
                        or a string constant.

HELP   This clause allows you to define a help message
            that can be displayed as a condition action

            HELP   is the clause keyword.

            string   is the character string that
                                you want to be displayed in the
                                message line as the result of a
                                specified condition. The
                                string and form_name options
                                are mutually exclusive.

            form_name   is the name of the predefined
                                form that you want to be
                                displayed as the result of a
                                specified condition. The
                                string and form_name options
                                are mutually exclusive.

   The DELETE, INSERT, and MODIFY actions are not available in
release 2.0.

Select Action

            This action causes data to be retrieved from the
            distributed database. The ADL uses the NDML SELECT
            command to retrieve the information. The syntax for
            the SELECT command is:

```
SELECT qualified name - col spec
       [ DISTINCT ]
       [ FROM table [ abbreviation ]         ]
       [ WHERE Predicate ]
       [ ORDER BY Col spec ( ASCENDING         ]
                           ( DESCENDING]
       [ | select action          ! ]
```

SELECT   is the action keyword

qualified name   the data you select is put int
                          the field item that you name
                          here The name must be
                          qualified with the path so that
                          it is unique

You must qualify the name enough to identify a unique
item field in the form hierarchy The name must be
enclosed in single quotes (i e   form item) This
is an exception to normal WDML syntax where single
quotes denote string constants In the ADL string
constants are denoted with double quotes and
qualified names are denoted with single quotes It
is optional to end the name with a semi colon If
you do not enter a semi-colon the system enters one
Whenever a repeating field is part of a qualified
name, it must include the subscript '1' (i e
 form(1) item) This refers to the current
occurrence of the field

Col spec   specifies the column name of
                          the data you are selecting from
                          the database table Only
                          columns from the table are
                          valid no quoted literal text
                          can be included and no
                          arithmetic can be performed on
                          the values selected before they
                          are displayed

DISTINCT

This clause is optional and specifies that
duplicate records selected from the database will
not be displayed   DISTINCT is the clause keyword

FROM

This clause specifies the table that contains the
data

FROM   is the clause keyword

table   is the full name of the table
                        that contains the data

abbreviation   allows you to define an
                        abbreviated name for the
                        database table   You can use
                        this name to refer to the table
                        throughout the application

WHERE

This clause allows you to specify criteria for
selection of the data from the database table

WHERE   is the clause keyword

Predicate   specifies the criterion for
                        selection of data   The syntax
                        for Predicate is

Predicate

```
      /                              \
     |  predicate AND predicate      |
     <  Operand Operator Operand     >
     |                               |
      \                             /
```

Operand

```
      /          \
     |  Col spec   |
        string
     <  number     >
     |  field name |
     |             |
      \          /
```

Operator

```
      /    .
     |   =   |
     |  '=   |
     .       .
     )   =   )
     .       .
     |   .=  |
        .
```

Col spec

```
     column
     table   column
     abbreviation   column
```

ORDER BY

This clause allows you to specify the order in
which the rows of data are displayed on the screen

ORDER BY   is the clause keyword

Col_spec   specifies the table column to use
                            as the sort key. More than one
                            sort key can be specified. You
                            can specify a sort direction
                            (ASCENDING OR DESCENDING) FOR
                            EACH SORT KEY. ASC is the
                            default.

SELECTs must be defined in the order in which the data are to be
presented on the screen. Nested SELECTs are grouped by use of
braces ({ }).

EXIT Action

        This action terminates the application.

SECTION 4

HOW TO CREATE AN APPLICATION DEFINITION

You create an application definition by writing ADL statements directly to a text file using any text editor.

4.1 Statement Format

ADL statements can be entered in free format. Free format means that keywords and numbers can be separated by 1 or more spaces. The ADL compiler treats tabs, comments, form feeds and carriage returns as spaces.

Application definition statements may be in any order.

The DOMAIN attribute options may be entered in any order.

Form field names used in the application definition syntax are pseudo qualified names in the form hierarchy. They are denoted by strings of the following type: 'form.item;'. The single quotes are required. For release 2.0, wherever a field is repeating, the qualified name must include the subscript "1" (i.e., 'form.item(1)'). This always refers to the current occurrence of the field.

4.2 Restrictions

Every application definition must begin with the CREATE APPLICATION statement.

There must be at least one space before and after every keyword in the syntax.

A field_name may only be omitted from the Location syntax in a PROMPT statement.

4.3 Abbreviations

Underscores in the ADL syntax indicate reserved words or portions of reserved words that are optional

4 4 <u>Including Comments</u>

You can include comments in a application definition by enclosing the comment text in (/*) and (*/). Comments are treated as spaces by the ADL compiler. For example:

CREATE APPLICATION application1 /* used to review salaries */

4.5 Reserved Words

This is an alphabetized list of the reserved words in the Application Definition Language.

ABOVE	COUNT	LEFT	RELATIVE
AND	CREATE	LOWER	REPORT
AP	CURSOR	MAX	RIGHT
APPLICATION	DESC	MAXIMUM	ROW
AS	DESCENDING	MIN	SELECT
ASCEND	DISPLAY	MINIMUM	SET
ASCENDING	DISTINCT	MUST	SIGNAL
AT	DOMAIN	NODUP	SIZE
AVERAGE	ENTER	NUM	SPACE
BACKGROUND	EXIT	NUMERIC	SPACES
BELOW	FILL	OF	STARTUP
BOTTOM	FORM	ON	SUM
BY	FROM	ORDER	SUMMARY
CENTER	HELP	OVERFLOW	TO
CHANGE	H	PAGE	TOP
COL	HORIZONTAL	PIC	UPPER
COLUMN	IN	PICK	VALUE
COND	ITEM	PICTURE	V
CONDITIONAL	KEY	PRESENT	VERTICAL
	KEYPAD	PROMPT	WHERE
			WINDOW
			WITH

APPENDIX A

STEPS FOR EXECUTING THE RAPID APPLICATION GENERATOR

Below is the procedure to use in invoking the Rapid
Application Generator for Release 2.0. This procedure assumes
that the NTM is up and running and the user is logged on to
IISS. Refer to the Terminal Operator Guide for the procedure
for logon to IISS. The following convention is used to document
this section.

● Text in angle brackets is to be replaced with appropriate
information by the user.

● Single upper case words enclosed in angle brackets
represent terminal keys (e.g. ‹ENTER›).

● Text in upper case is to be entered as shown.

1 On the IISS Function Screen, fill in the following items:

FUNCTION: SDAPPGENER   Press ‹ENTER›

2 On the Rapid Application Generator screen, fill in the
following items:

Application Definition File Name   Press ‹ENTER›

This file may reside anywhere as long as the complete
filename is given. It is the file containing the
application definition.

3 After the Rapid Application Generator has completed the
following files exist in the NTM environment directory (all
prefixed with the application name as given on the CREATE
APPLICATION line of the application language file. For
example, if the application name is UPDATEDB

UPDATEDBC C   (generated C code for form processing)
UPDATEDBX PRC   (generated Cobol code for the NDML)
UPDATEDB H   (a header file to be used by UPDATEDBC C)
UPDATEDB WRN   (lists any errors where the External Schema
                 and Presentation (Form) Schema item mapping
                 may cause truncation)
fd   (form definition files)

It is best to use 8 characters for application names.
These, together with the SD prefix make up the 10
characters required for the name used when executing the
generated application program.

4   Now press the <QUIT> key twice to exit the Rapid
Application Generator and the IISS and return to the
operating system command level.

5   Run the NDML precompiler on the UPDATEDBX.PRC file. Follow
the NDML Programmer's Guide.

6   Output from the NDML precompile consists of the following
files (the exact file names can be found by looking at the
output messages from the NDML precompile):

generated application program (with NDML statements
converted to correct language (Cobol or Fortran))

one or more RP-Sub process files

one or more CS-ES subroutine files

an RP-main processor

7   All the above files must be compiled following the
procedure given in the NDML Programmer's Guide.

8   Compile the generated UPDATEDBC.C program.

9   Link the generated RP main process and subprocess using
LNKORP as outlined in the NDML Programmer's Guide

10   Link the generated application object, the CS-ES object(s)
and the generated C application program into one
executable For this example, the executable is named
UPDATEDB

11   Update the UI database This step shows how to do this
using ORACLE on the VAX

```
$ DELETE SEL* DAT.*
$ UFI
username   username
password   password
```

Enter the following line

INSERT INTO ROLAPP VALUES (`‹role in capital letters›`,`SD‹application name padded to 8 characters with Z's›`); EXIT

12   Update the NTM tables. On the Vax this requires editing the following .DAT files.

$ EDIT/EDT ACTTBL.DAT

Insert new lines as follows:

‹application name padded to 8 characters with Z s 1
‹RP-Main process name padded to 8 characters with Z s 1

$ EDIT/EDT APITBL.DAT

Insert new lines as follows:

SD‹application name padded to 8 character with Z s›T1V1
GR‹RP-Main Bxxxx process name padded to 8 characters with
Z's›T1V1

$ EDIT/EDT APTTBL.DAT

Insert new lines as follows:

‹application name padded to 8 characters with
Z s›05990010320000010
‹RP-Main name padded to 8 characters with
Z s›99990010120001130

Perform steps 13 through 16 to update the UI database

13   Return to the IISS and on the IISS Function Screen fill in the item as follows

FUNCTION: SDDEFINEAP

14   On the Define An Application form, fill in the item as follows:

APPLICATION: SDUPDATEDB

This name must match the executable name in step 10 and must be a total of 10 characters including the SD prefix

A-3

15   On the Define New Application form, fill in the items as
      follows:

      DESCRIPTION:
      HOST:
      CLUSTER:
      Press ‹ENTER›

      Host and Cluster are variable and depend on the
      application.

16   Press ‹QUIT› to return to the IISS Function Screen.

      To run the generated application program, perform step 17.

17   On the IISS Function Screen, fill in the item as follows:

      FUNCTION: SDUPDATEDB   Press ‹ENTER›

END

8-87

DTIC